October 2, 2025

6.S894 Accelerated Computing

Lecture 5: Memory, Overlapping Compute & I/O

Jonathan Ragan-Kelley

L1 SRAM

128 KB per-SM

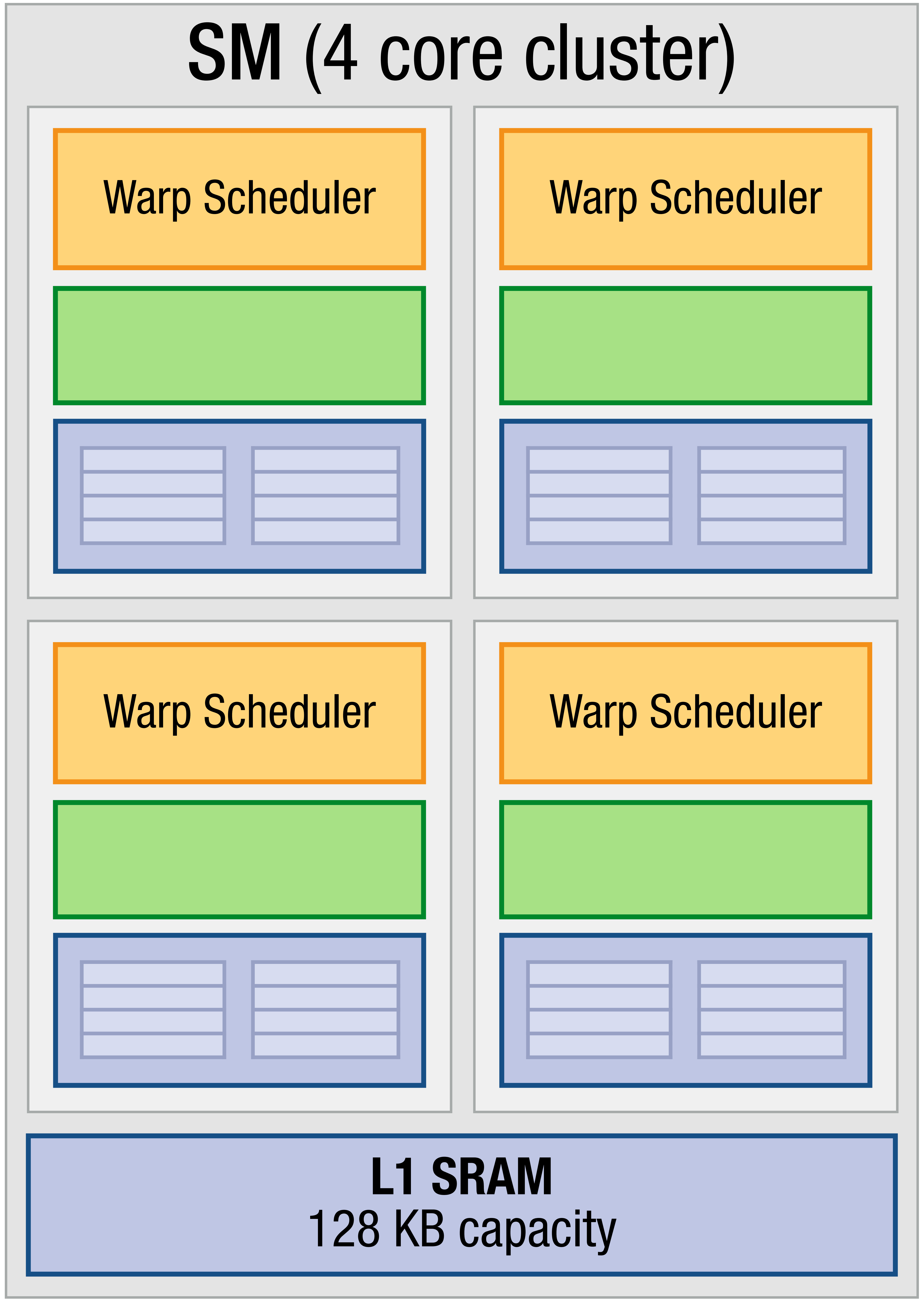
($\times$ 48 SMs = 6 MB)

128 bytes / cycle / SM

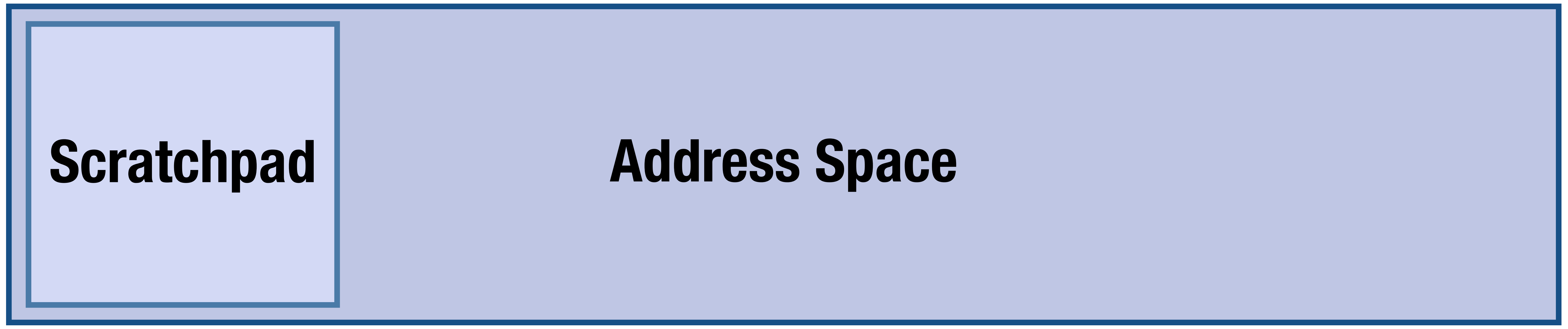
↳ 1 warp-wide ld/st

(Per-core: 1 every 4 cycles)

$\times$48 SMs $\times$ 2.18GHz = 13.4 TB/s



L1 SRAM: also used as **explicit scratchpad**



Each block (SM) only sees
its own scratchpad

Tradeoffs: scratchpads vs. caches

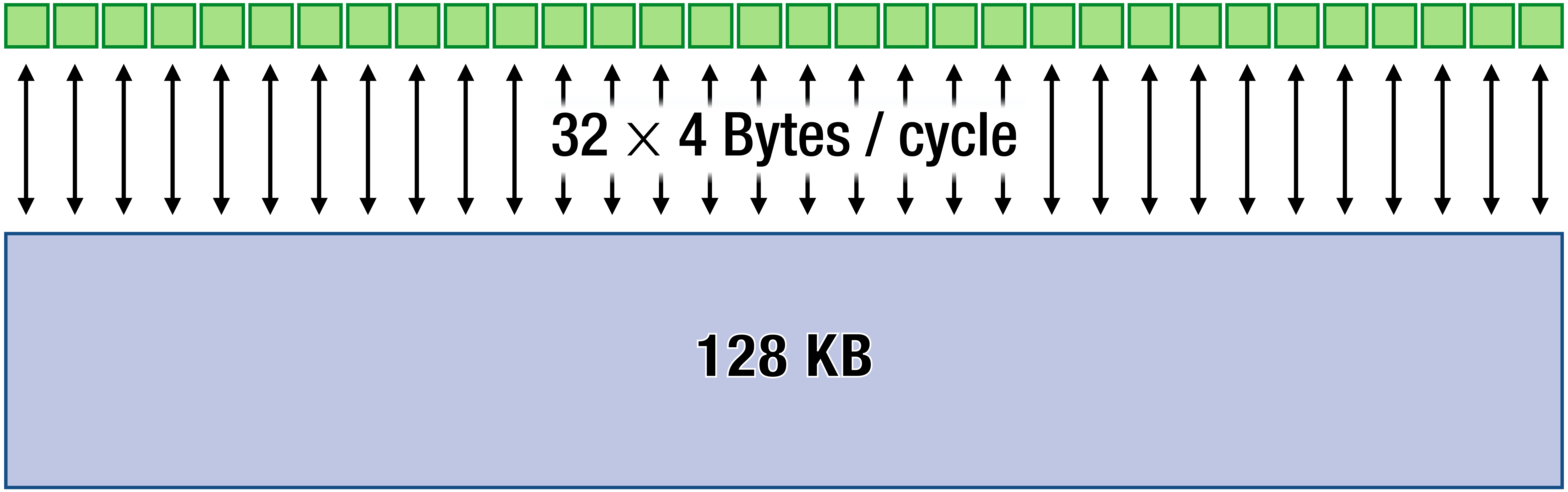
+ No tag / lookup overhead
+ Predictable / controllable
+ No need for coherence
+ Read >1 line / cycle

– Programming effort

– Software depends on size

How is the L1 SRAM built?

How can it deliver 128 B / cycle? **Parallelism!**



32 parallel **"banks"** (addresses **striped** % 32 × 4 Bytes)

gather

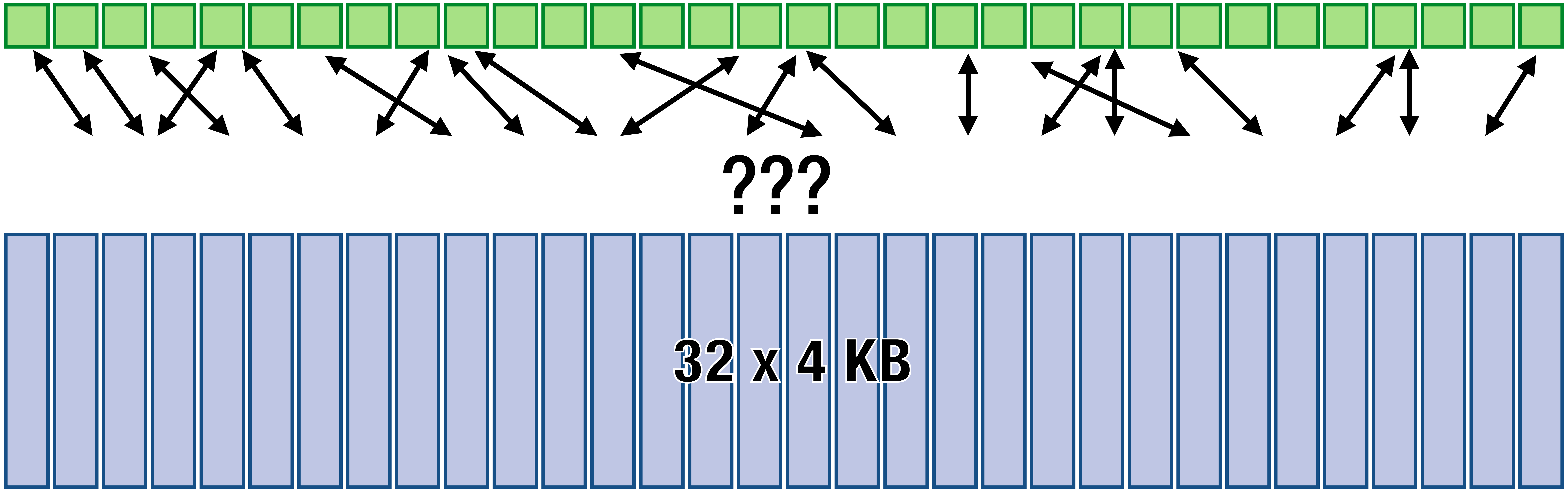
32 × 4 Bytes / cycle

…
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
EU
…

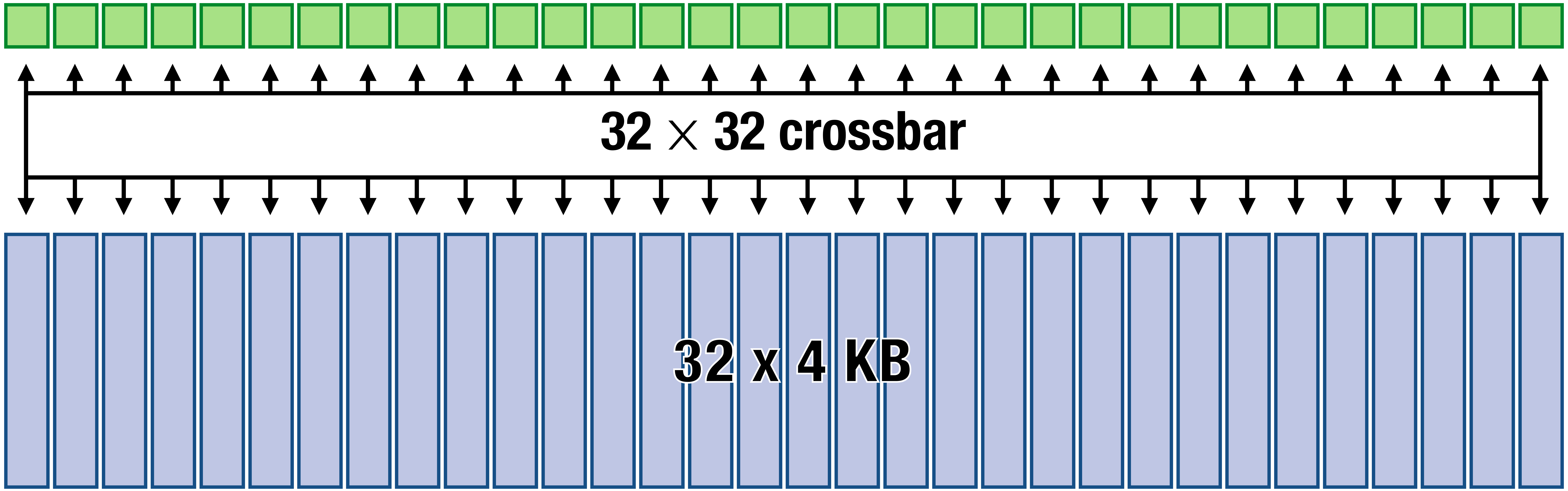
32 × 4 Bytes / cycle

scatter

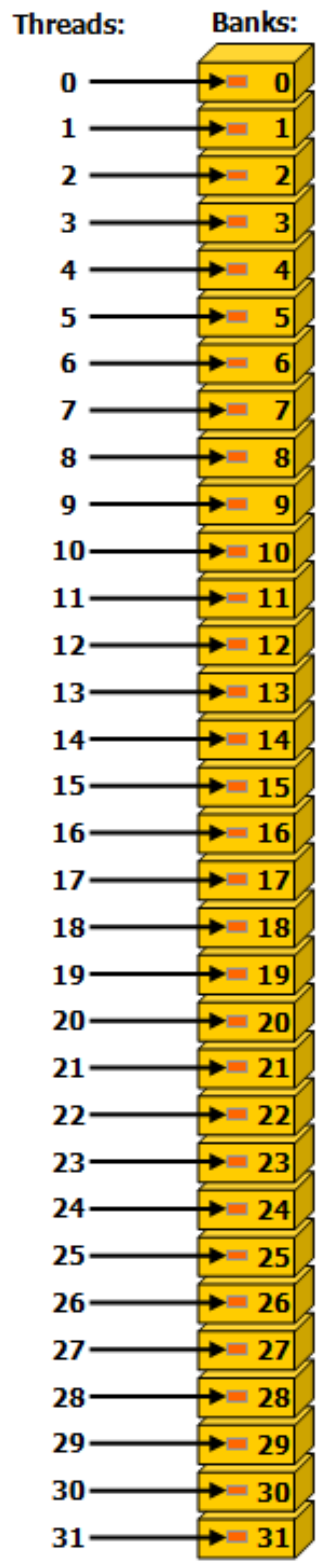
How can we handle **gather/scatter** in the **L1 SRAM?**

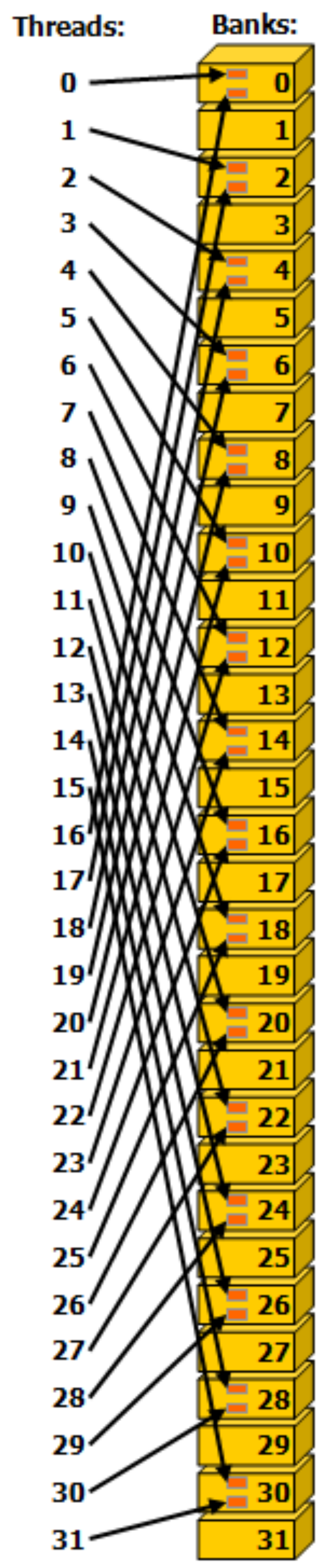


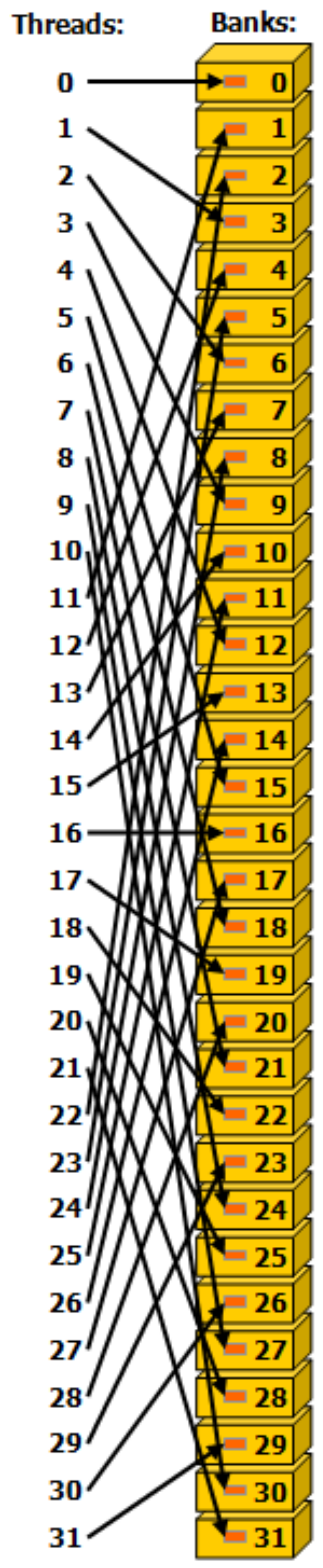
How can we handle **gather/scatter** in the **L1 SRAM?** All-to-all **crossbar!**

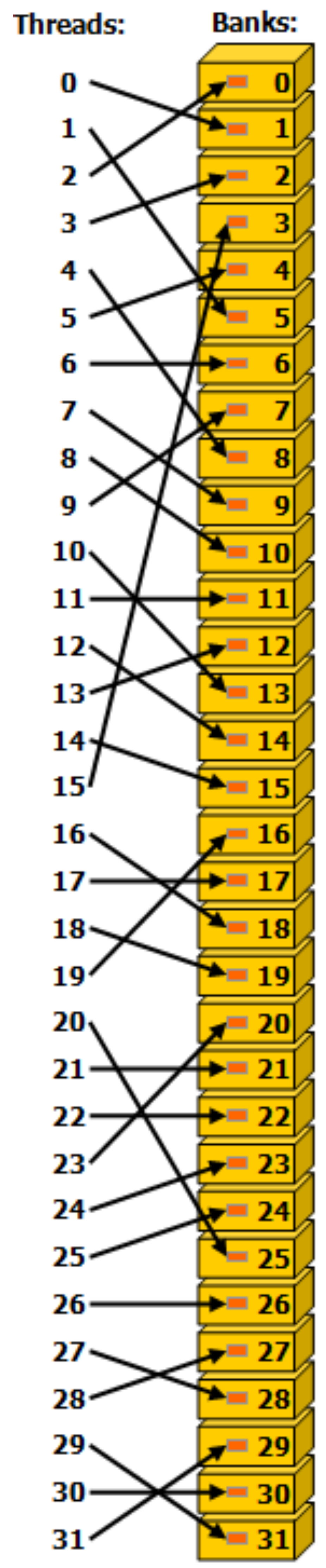


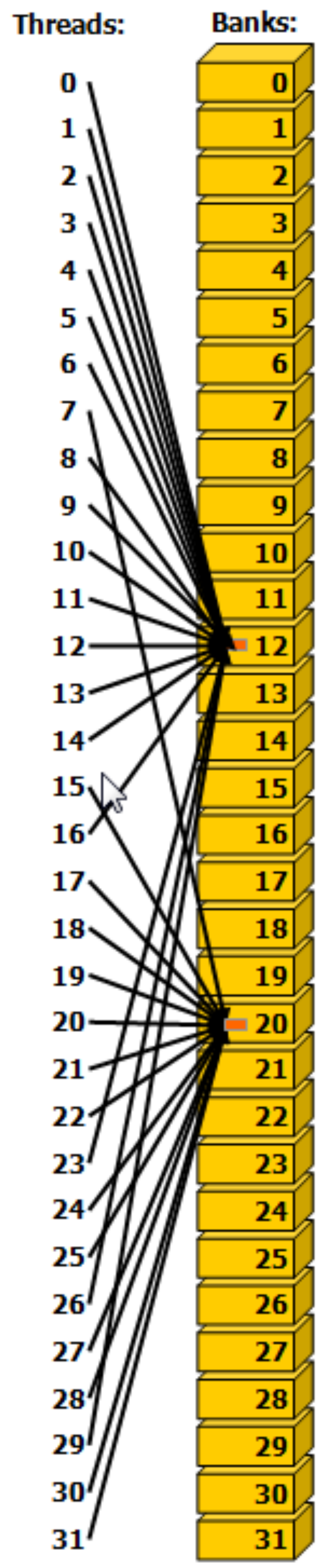
L1 bank conflicts

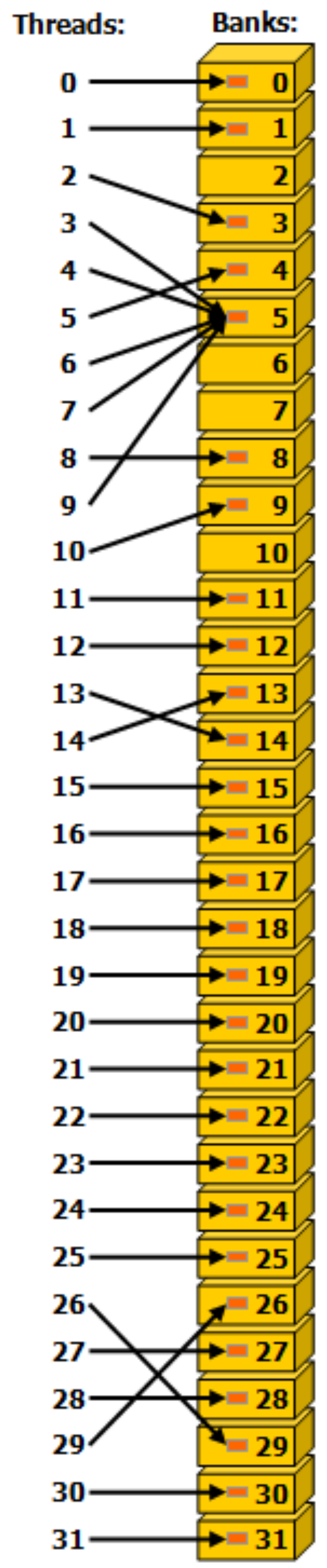


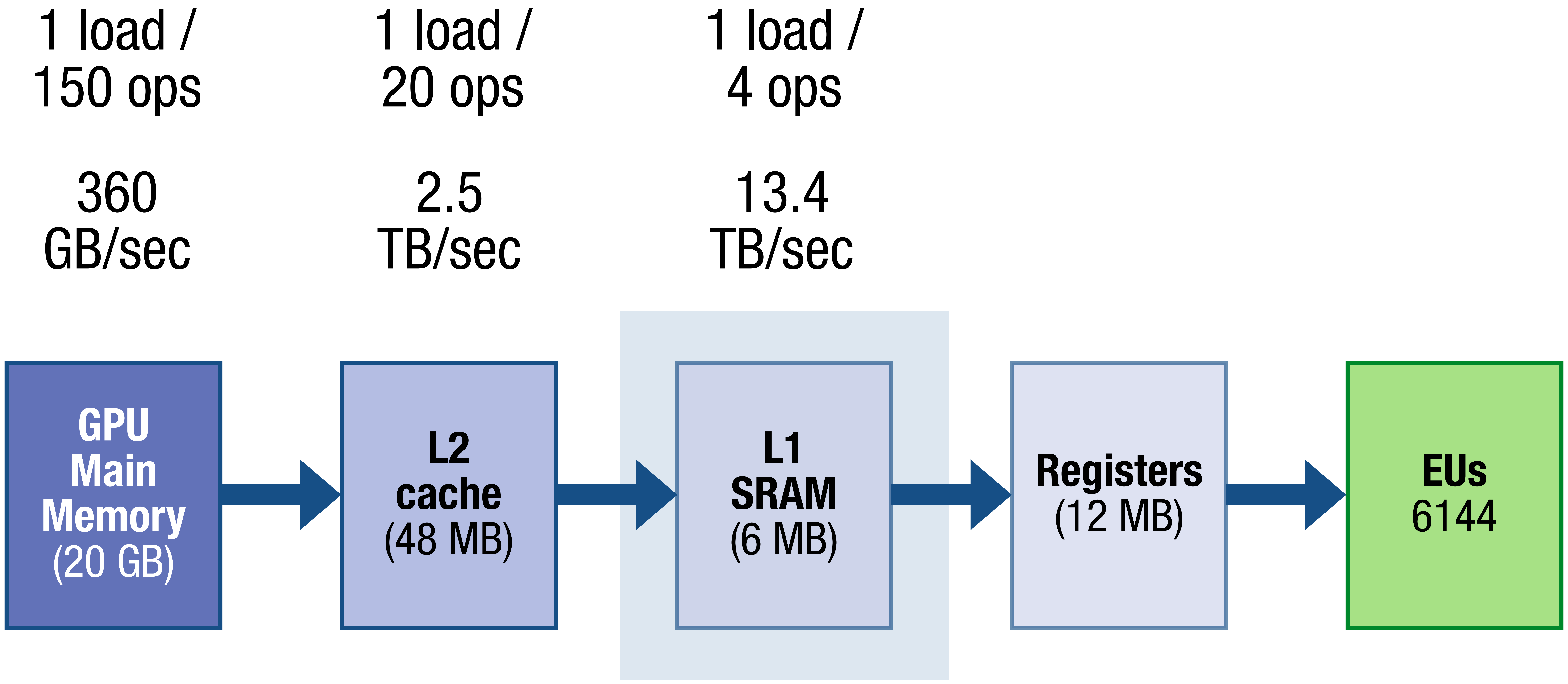












high **bandwidth**, limited **capacity**

high clocks & wide interface

aggregate large transactions for DRAM

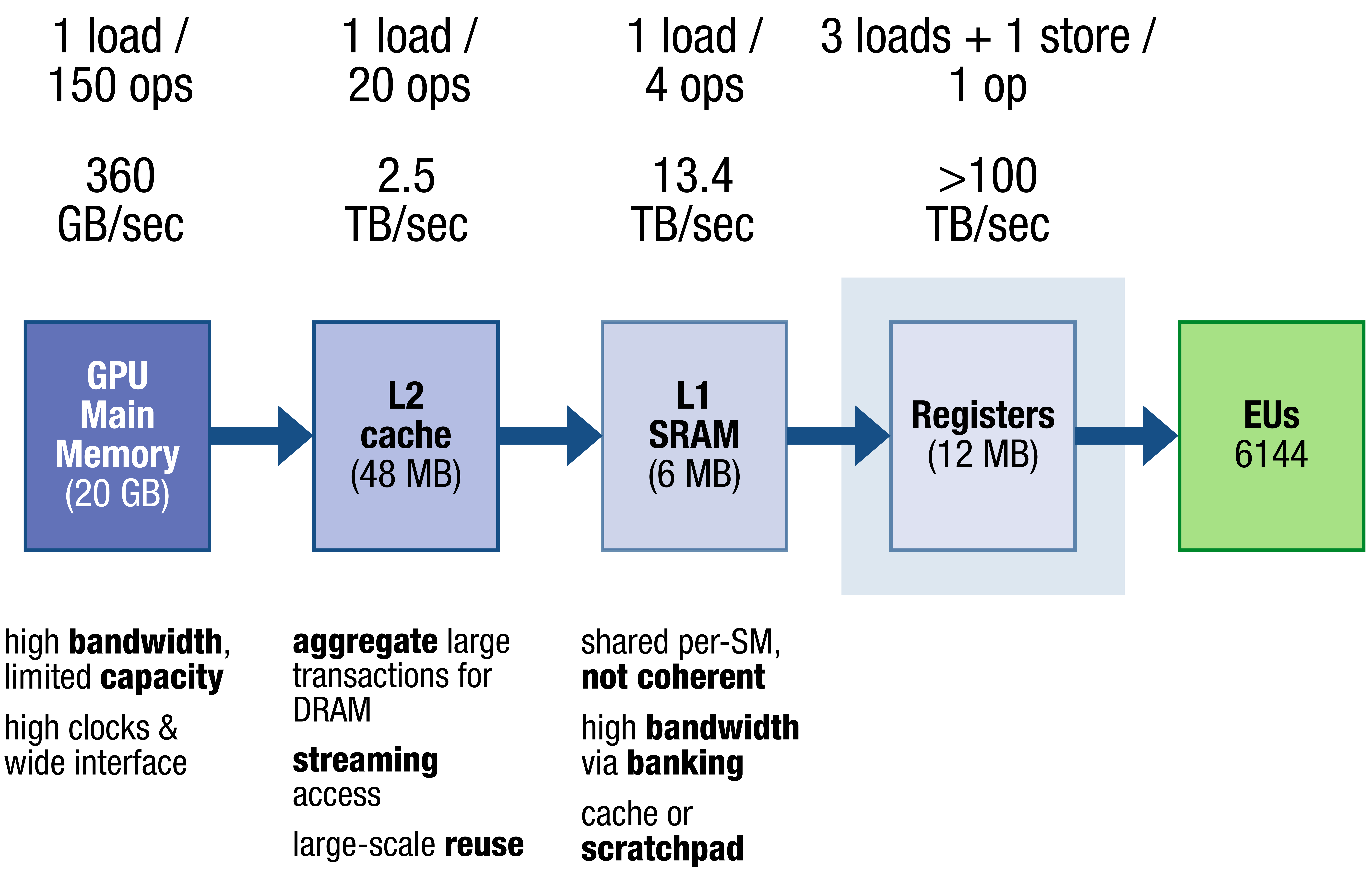
streaming access

large-scale **reuse**

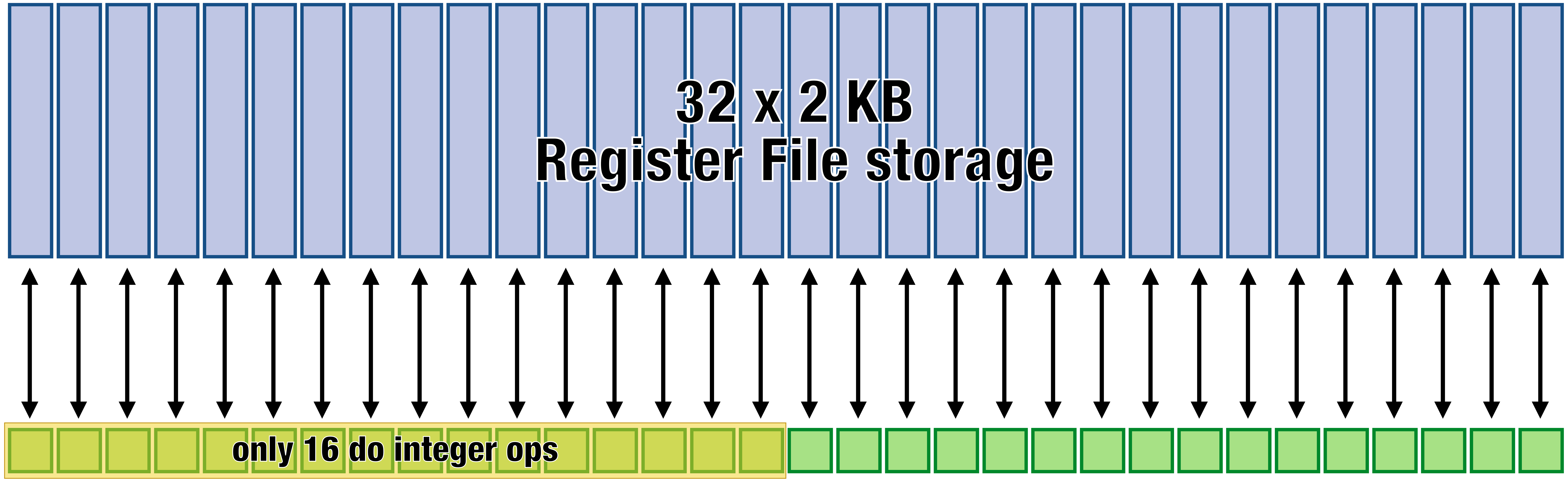
shared per-SM, **not coherent**

high **bandwidth** via **banking**

cache or **scratchpad**


1 load / 150 ops
360 GB/sec
1 load / 20 ops
2.5 TB/sec
1 load / 4 ops
13.4 TB/sec
3 loads + 1 store / 1 op
>100 TB/sec
GPU Main Memory (20 GB)
L2 cache (48 MB)
L1 SRAM (6 MB)
Registers (12 MB)
EUs 6144
high **bandwidth**, limited **capacity**
high clocks & wide interface
aggregate large transactions for DRAM
streaming access
large-scale **reuse**
shared per-SM, **not coherent**
high **bandwidth** via **banking**
cache or **scratchpad**

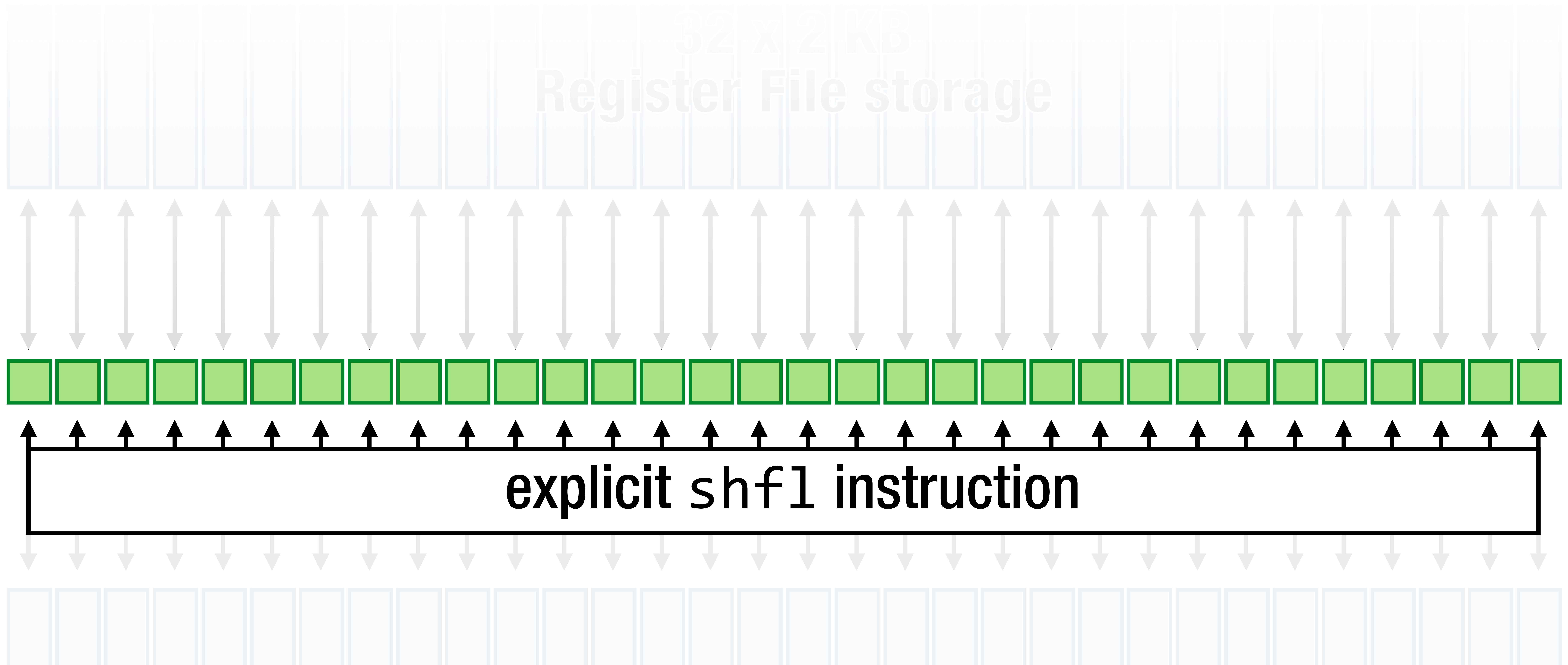
Register file: parallel banks per-lane



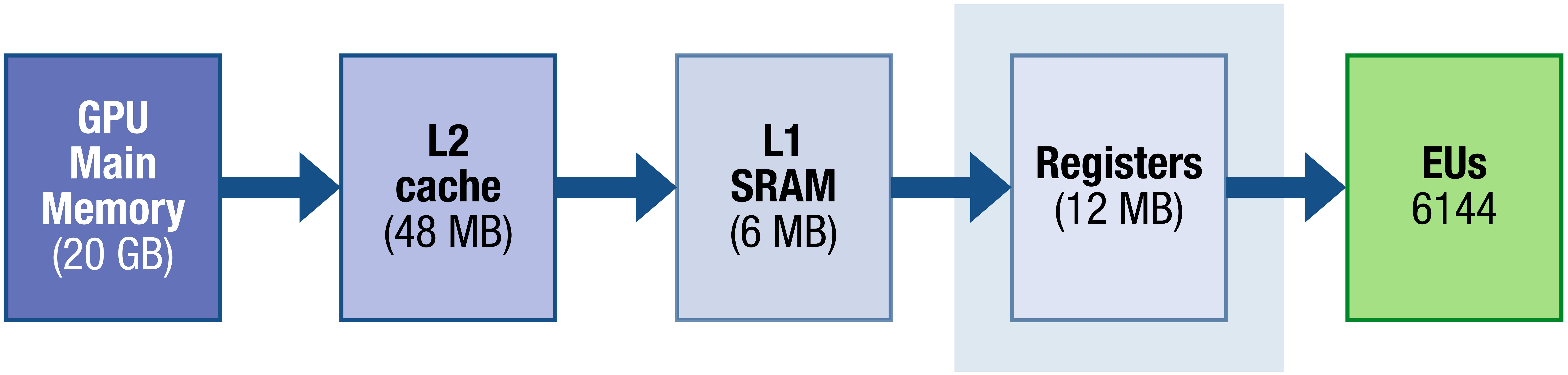
no `r[3]` (or `r[i]`), only `r3`

explicit `shfl` instruction

Data can be **shuffled between lanes** using the **L1 crossbar**



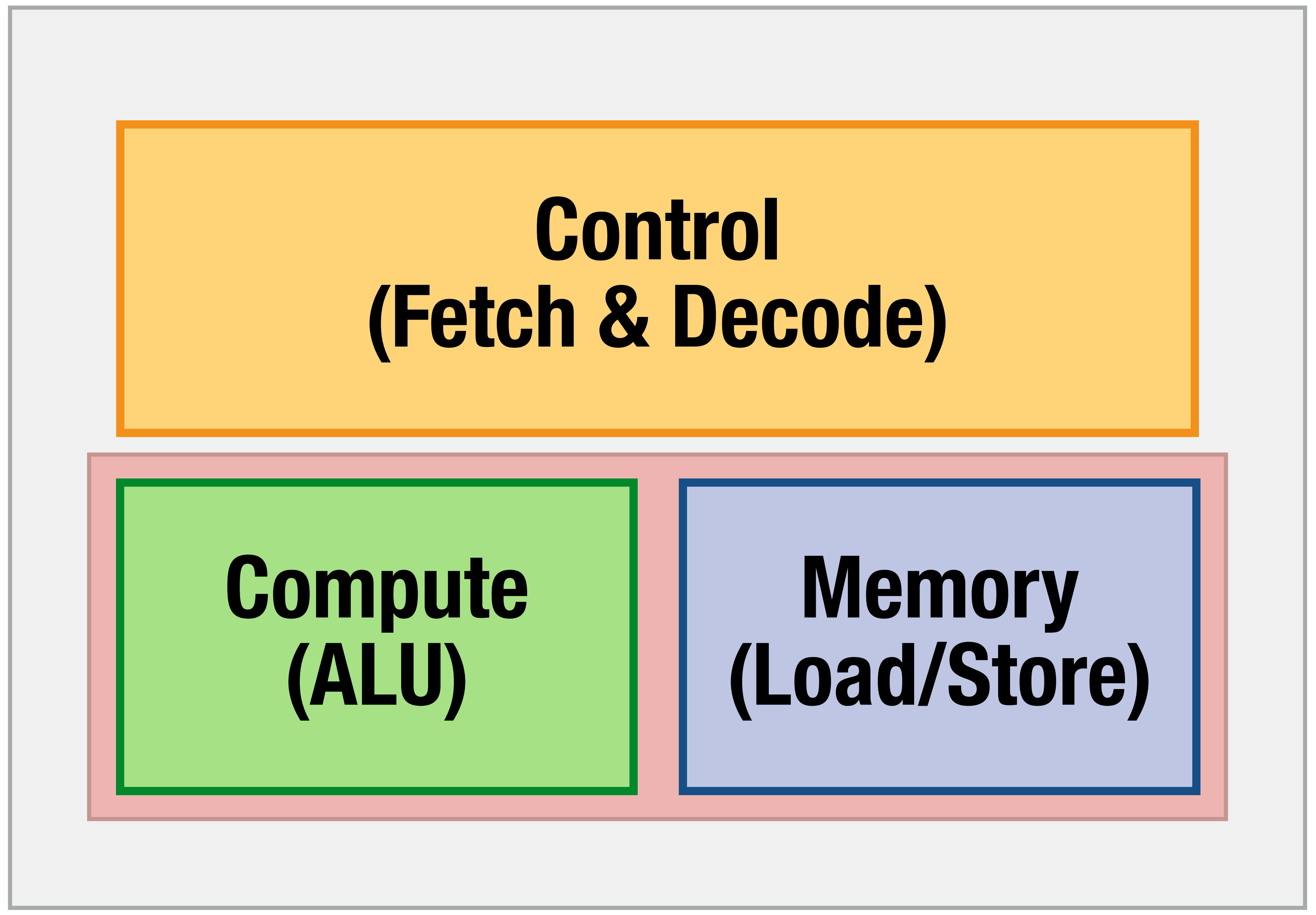
1 load / 150 ops	1 load / 20 ops	1 load / 4 ops	3 loads + 1 store / 1 op	
360 GB/sec	2.5 TB/sec	13.4 TB/sec	>100 TB/sec	



high **bandwidth**, limited **capacity**	**aggregate** large transactions for DRAM	shared per-SM, **not coherent**	**banked** per-lane
high clocks & wide interface	**streaming** access	high **bandwidth** via **banking**	no dynamic **indexing**, either across or within lanes
	large-scale **reuse**	cache or **scratchpad**	“infinite” BW

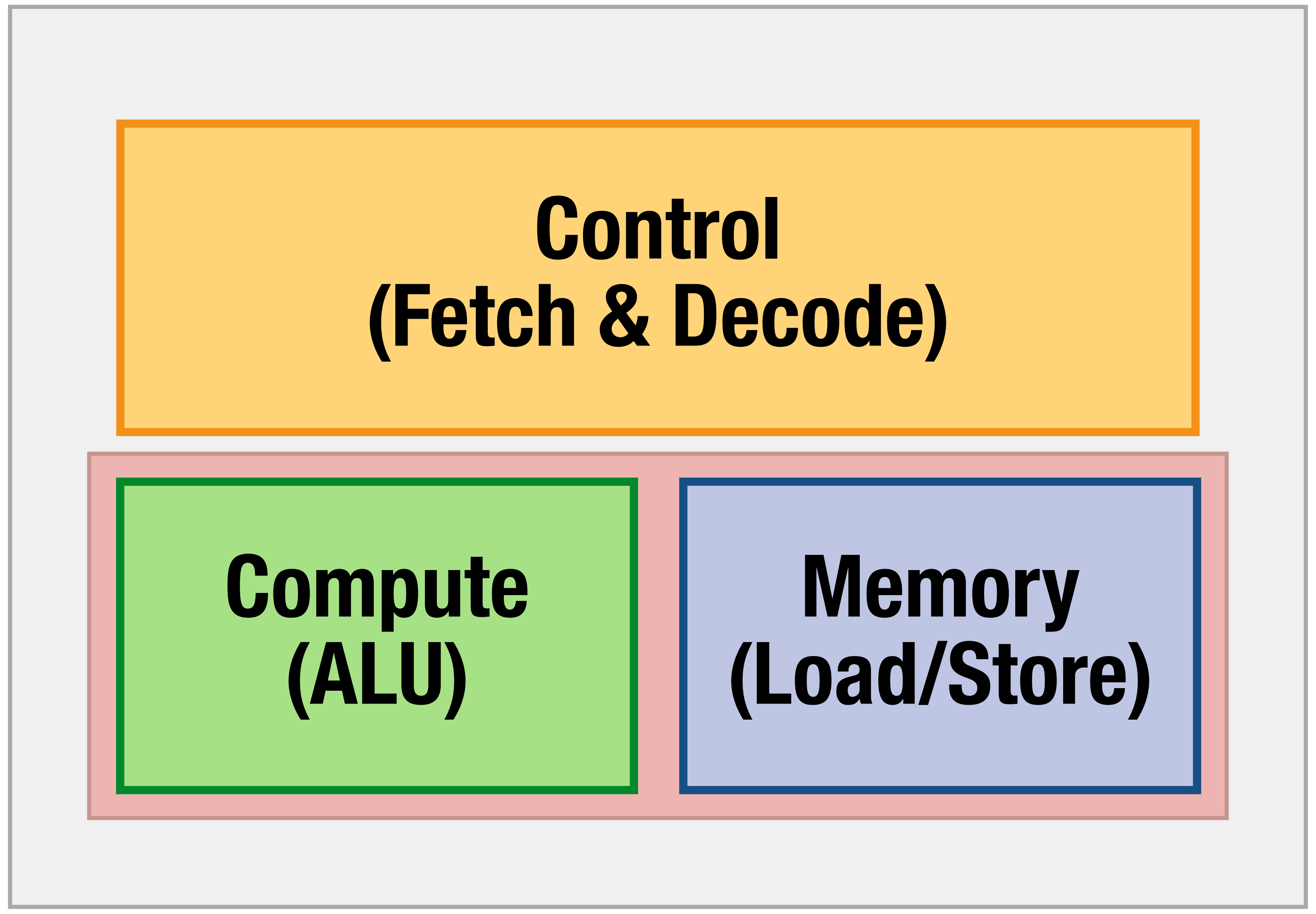
Overlapping compute & I/O

Key facets of a processor:
Control, Compute, Memory



Goal: fully utilize
both resources

Use compute & memory **in parallel**

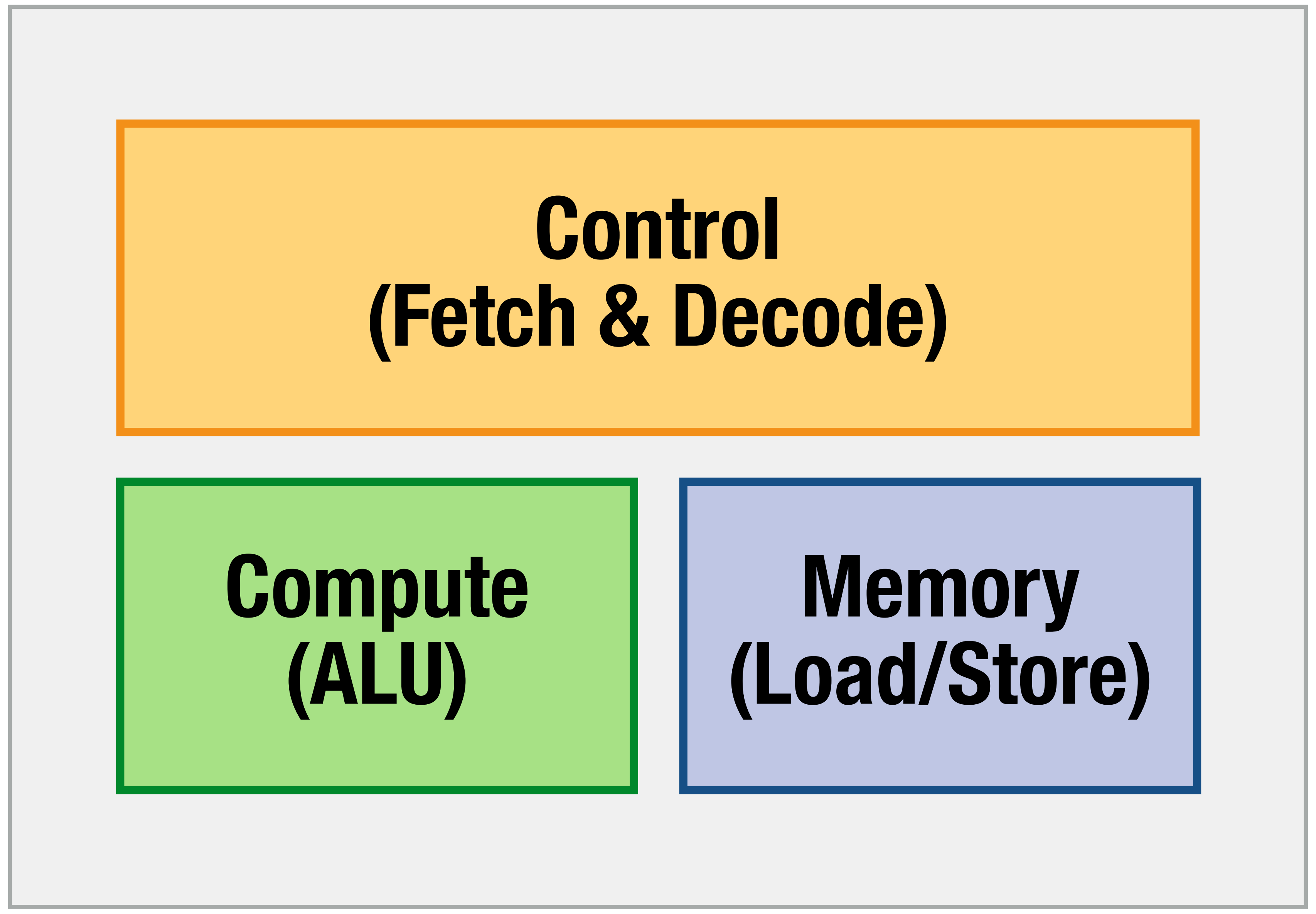


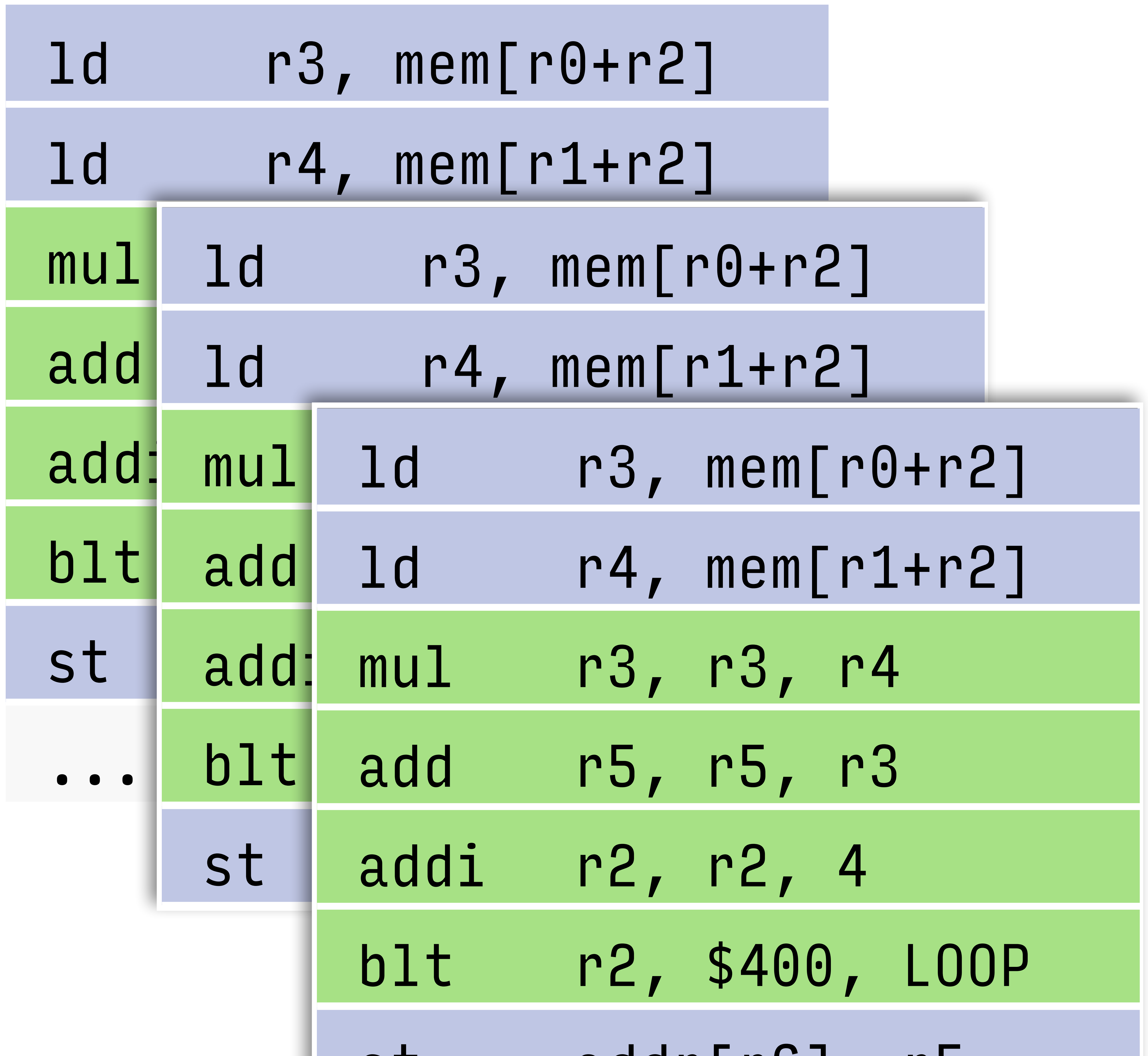
Goal: fully utilize
both resources

Approach 1: “CPU-style”
wide issue, out-of-order
parallelism **within** instruction stream

Approach 2: “GPU-style”
multithreading
parallelism **across** instruction streams

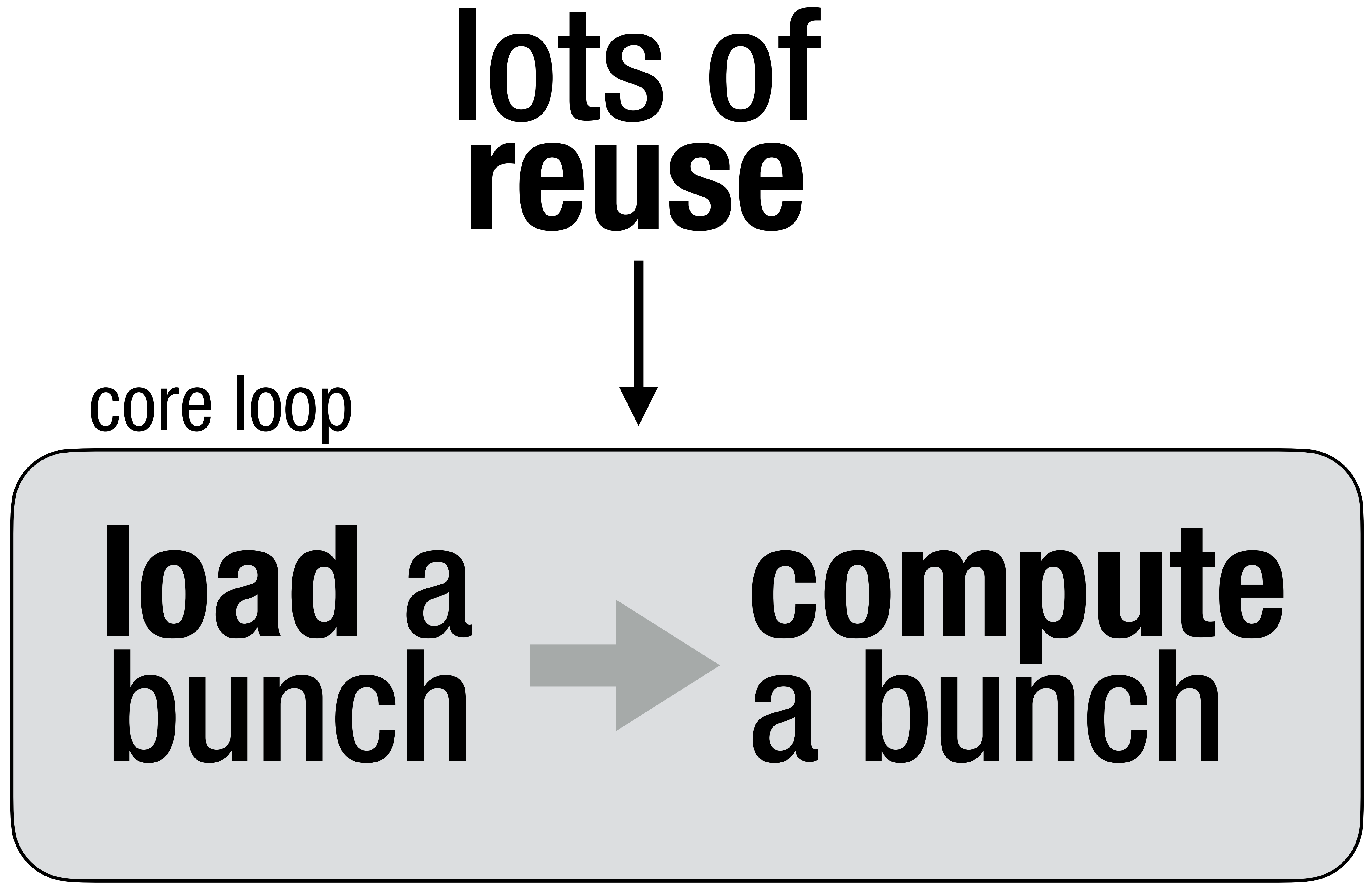
Both strategies work for common workloads





```
ld      r3, mem[r0+r2]
ld      r4, mem[r1+r2]
mul     r3, r3, r4
add     r5, r5, r3
addi    r2, r2, 4
blt     r2, $400, LOOP
```

What about workloads like **matrix multiplication?**



```
// load into scratchpad
for i,j:
  load A,B global → scratch

for each microtile:
  // load into registers
  for i,k:
    load A scratch → reg
  for k,j:
    load B scratch → reg

  // compute!
  for i,j,k:
    compute C += A*B
```

Front End
Zeroing Idioms
6-Way Rename / Dispatch
Move Elimination
Register Alias Tables
6 Micro-Ops
8 Micro-Op Retire
Reorder Buffer (320 entry)
Scheduler (24 entry)
Scheduler (24 entry)
Scheduler (24 entry)
Scheduler (24 entry)
Non-Scheduling Queue (64 entry)
Scalar Integer Registers and Forwarding Network
Flags Register File (238 entry)
Integer Register File (224 entry)
Scheduler (32 entry)
Scheduler (32 entry)
FP/Vector Registers and Forwarding Network
AVX-512 Mask Register File (68 entry)
FP/Vector Register File (192x 512-bit entry)
ALU
Branch
ALU
AGU
ALU
AGU
ALU
Branch
AGU
FMA
FADD
FStore
F2I
FMA
FADD
FStore
F2I
L2 Cache
1 MB 8-Way
Execution Engine
Load Queue (136 entry)
Store Queue (64 entry)
3x64-bit or 2x256-bit loads/cycle
2x64-bit 1x256-bit store/cycle
DTLB
72 entry L1
3072 entry L2
L1 Data Cache
32 KB 8-Way
32 Bytes/Cycle
Miss Address Buffer (MAB)
>= 24 entry?
Memory Subsyste

Zen 4 core

diagram by
Chips & Cheese

What about a **throughput processor** (GPU)?

- single issue
- in-order
- “RISC”

→

more manual & explicit management of **overlapping**

↳ recurs at **many levels**

Problem 1: load / store instructions are **asynchronous & long-latency**

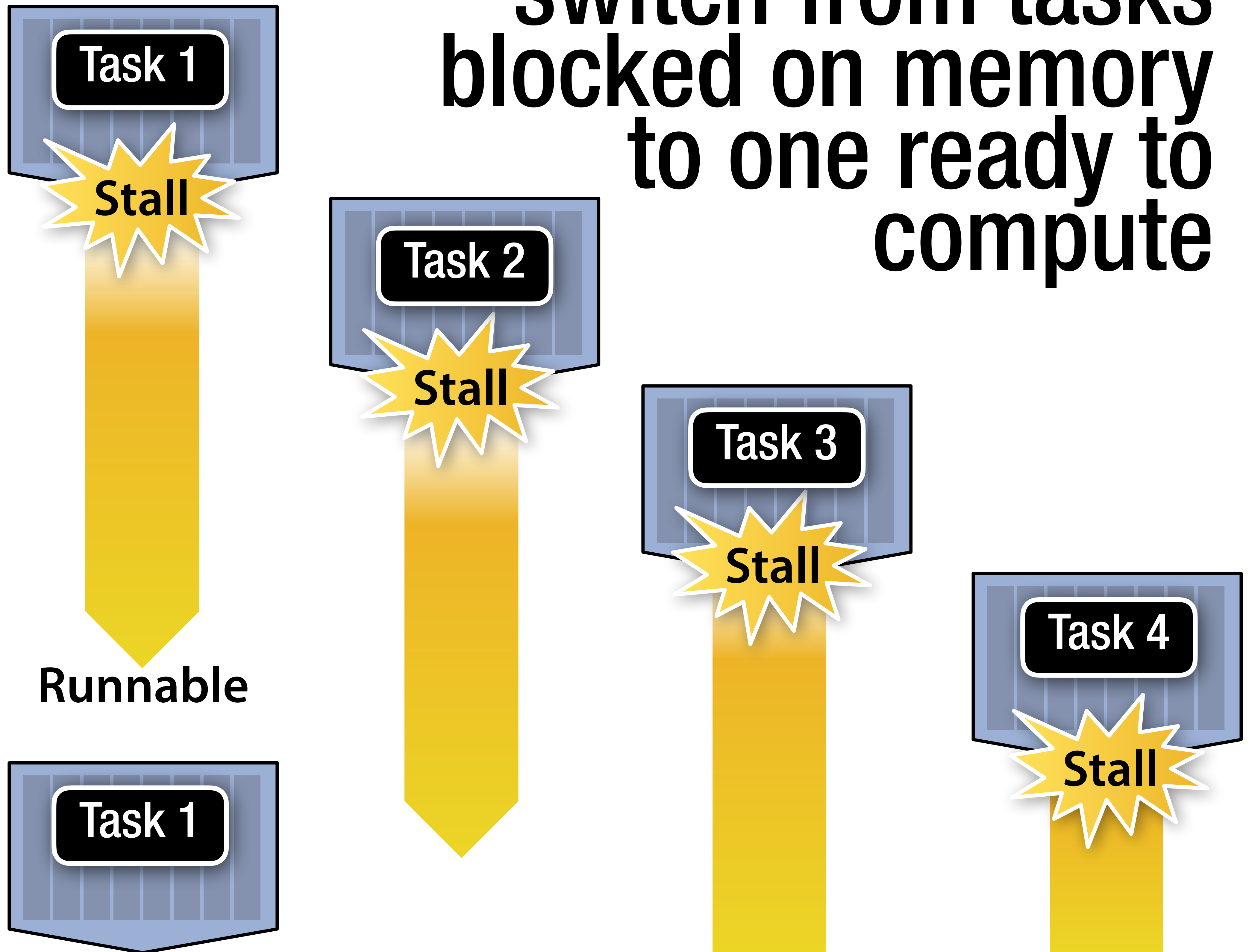
Solution 1: ILP

hoist loads early to avoid blocking

```
ld          ld
fma         ld
ld    →     ld
fma         fma
ld          fma
fma         fma
…           …
```

Solution 2: multithreading

switch from tasks blocked on memory to one ready to compute



Problem 2: load / store instructions waste **issue slots**

Solution: bulk load / store instructions
e.g.,“vectorized” ld / st

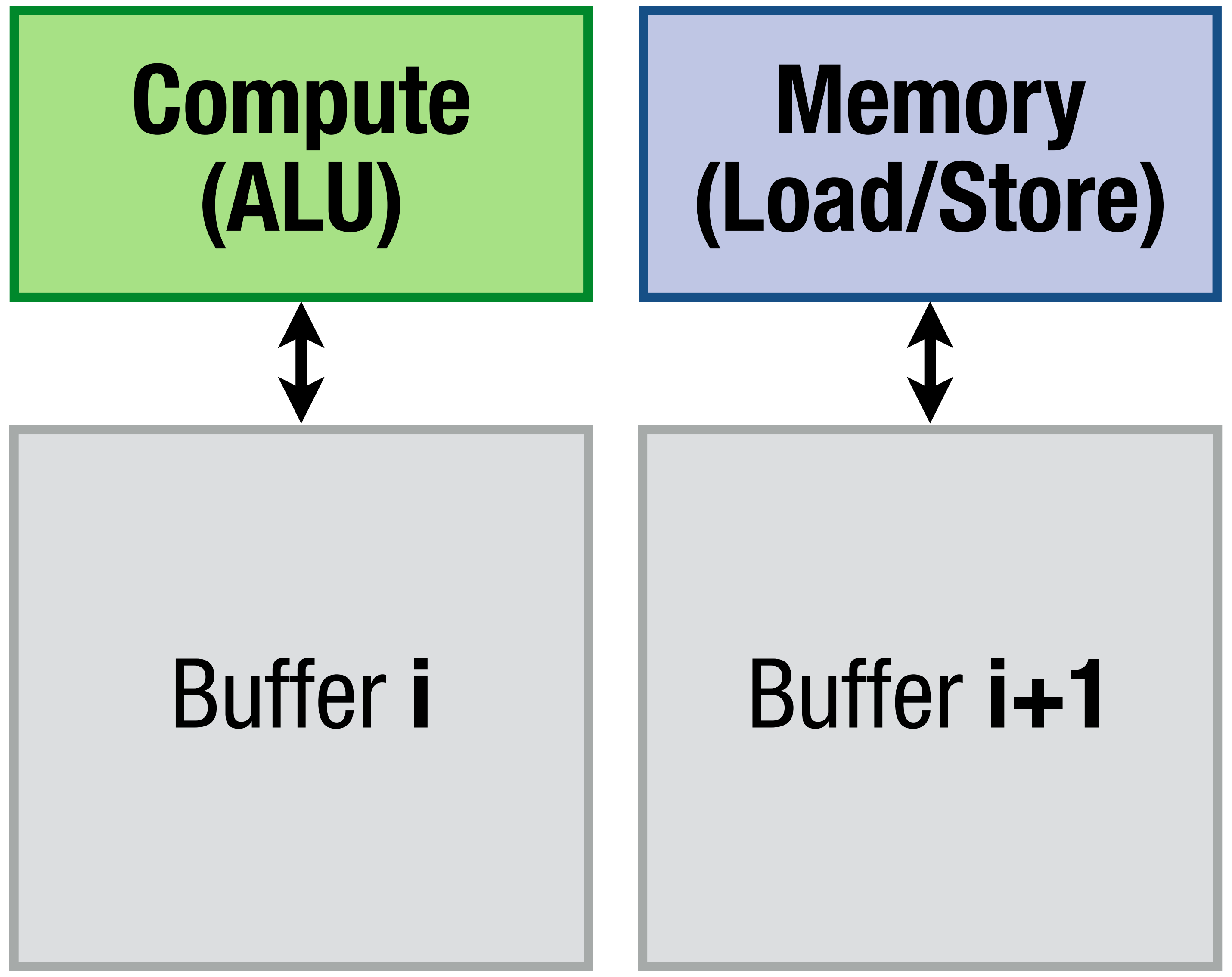
```
ld.f32
ld.f32
ld.f32
ld.f32
fma
fma
fma
fma
…
```

$\rightarrow$

```
ld.v4.f32
fma
fma
fma
fma
…
```

Problem 3: overlapping compute with loading to the **scratchpad**

Solution:
asynchronous fetch &
double-buffering



Implementation approach:
warp specialization

```
if threadIdx.y < 4:
  // load into scratchpad
  for i,j:
    load next A,B → scratch
else:
  // compute!
  for i,j,k:
    compute C += A*B
sync & swap buffers…
```